Topic-Oriented Collaborative Crawling

Chiasen Chung Charles L. A. Clarke

School of Computer Science, University of Waterloo, Canada
c3chung,claclark@plg.uwaterloo.ca

ABSTRACT

A major concern in the implementation of a distributed Web crawler is the choice of a strategy for partitioning the Web among the nodes in the system. Our goal in selecting this strategy is to minimize the overlap between the activities of individual nodes. We propose a topic-oriented approach, in which the Web is partitioned into general subject areas with a crawler assigned to each. We examine design alternatives for a topic-oriented distributed crawler, including the creation of a Web page classifier for use in this context. The approach is compared experimentally with a hash-based partitioning, in which crawler assignments are determined by hash functions computed over URLs and page contents. The experimental evaluation demonstrates the feasibility of the approach, addressing issues of communication overhead, duplicate content detection, and page quality assessment.

Categories and Subject Descriptors

H.3.4 [**Information Systems**]: Information Storage and Retrieval—*Systems and Software*

General Terms

Measurement, Performance, Experimentation

Keywords

Web crawling, Distributed systems, Text categorization

1. INTRODUCTION

A *crawler* is a program that gathers resources from the Web. Web crawlers are widely used to gather pages for indexing by Web search engines [12, 23], but may also be used to gather information for Web data mining [16, 20], for question answering [14, 28], and for locating pages with specific content [1, 9].

A crawler operates by maintaining a *pending queue* of URLs that the crawler intends to visit. At each stage of the crawling process a URL is removed from the pending queue, the corresponding page is retrieved, URLs are extracted from this page, and some or all of these URLs are inserted back into the pending queue for future processing. For performance, crawlers often use asynchronous I/O to allow multiple pages to be downloaded simultaneously [4, 6] or are structured as multithreaded programs, with each thread executing the basic steps of the crawling process concurrently with the others [23].

Executing the crawler threads in parallel on a multiprocessor or distributed system further improves performance [4, 10, 23]. Implementation on a distributed system has the potential for allowing wide-scale geographical distribution of the crawling nodes, minimizing the competition for local network bandwidth. A *distributed crawler* may be structured as a group of n *local crawlers*, with a single local crawler running on each node of an n-node distributed system. Each local crawler may itself be a multithreaded program, maintaining its own pending queue and other data structures, which are shared between the locally executing threads. A local crawler may even be a parallel program executing on a cluster of workstations, with centralized control and a global pending queue. In this paper, we use the term "distributed crawler" only in reference to systems in which the data structures and system control are fully distributed.

Local crawlers cannot operate entirely independently. Collaboration is necessary to avoid duplicated effort in several respects. The local crawlers must collaborate to reduce or eliminate the number of resources visited by more than one local crawler. In an extreme case, with no collaboration, all local crawlers might execute identical crawls, with each visiting the same resources in the same order. Overall, the distributed crawler must partition the Web between the local crawlers so that each focuses on a subset of the Web resources that are targeted by the crawl.

Collaboration is also needed to identify and deal with duplicated content [5, 13, 23]. Often multiple URLs can be used to reference the same site (`money.cnn.com`, `cnnfn.com`, `cnnfn.cnn.com`). In some cases, large sets of inter-related pages will be encountered repeatedly during a crawl, and the crawler should avoid visiting each copy in its entirety. For example, many copies of the Sun Java JDK documentation can be found on the Web. If the results of a crawl are used in a Web search system, the user might only be presented with the most authoritative copy of a resource, such as the copy of Java documentation on the Sun Website. Similarly, Web-based question answering [14] depends on independence assumptions that duplicated content in-

CIKM'02, November 4–9, 2002, McLean, Virginia, USA.

validates. To recognize duplicated content, a local crawler might compute a hash function over contents of each Web page [13, 23]. In a distributed crawler, sufficient information must be exchanged between the local crawlers to allow duplicates to be handled correctly.

Collaboration may be necessary to effectively implement URL ordering algorithms. These algorithms attempt to order the URLs in the pending queue so that the most desired pages are retrieved first. The resulting order might reflect the expected quality of the pages or the need to refresh a local copy of important pages whose contents are known to change rapidly. Cho et al. [12] compare several approaches for ordering URLs within the pending queue. They introduce an ordering based on the *pagerank* measure [4], which is a measure of a page's expected quality, and compare it with a breadth-first ordering, a random ordering, and an ordering based on a simple backlink count. Their reported experiments demonstrate that the pagerank ordering is more likely to place important pages earlier in the pending queue.

Given a page P, its pagerank $R(P)$ may be determined from the pagerank of the pages $(T_1, T_2, ...T_k)$ that link to it:

$$R(P) \;=\; (1-d) + d\left(\sum_{i=1}^{k} R(T_i)/c_i\right)$$

where d is a damping factor whose value is usually in the range 0.8 to 0.9, and where c_i is the outdegree of T_i — the total number of pages with links from T_i. Given a set of pages, their pagerank values may be computed by assigning each page an initial value of 1 and then iteratively applying the formula above. In the context of a crawler, pagerank values may be estimated for the URLs in the pending queue from the backlink information provided by pages that have already been retrieved. Because of the dependence on backlink information, local crawlers may need to exchange this information in order to accurately estimate pagerank values within their local pending queues [10].

Ideally, the local crawlers would operate nearly independently, despite the need for collaboration. The amount of data exchanged between local crawlers should be minimized, preserving network bandwidth for the actual download of the targeted Web resources. For scalability, the total amount of data sent and received by a local crawler should be no more than a small constant factor larger than the amount of received data that is actually associated with the crawler's partition. Synchronization between local crawlers, in which the operation of one local crawler may be delayed by the need to communicate with another local crawler, should be avoided. Avoiding synchronization is especially important when local crawlers are geographically distributed, and network or node failures may disrupt communication.

Finally, it is desirable for the output of each local crawler to be usable as an independent subcollection. For example, if the crawl is being generated for use by a distributed application, such as a distributed search engine, it may be possible to transfer data directly from the source nodes in the distributed crawler to destination nodes in the distributed application, without ever centralizing the data. If the output of a local crawler is to be used as an independent subcollection it should exhibit internal "cohesion", with a link graph containing densely connected components, allowing pagerank and other quality metrics to be accurately estimated [2, 4, 27].

In this paper we present X4, a topic-oriented distributed crawling system. In X4, the Web is partitioned by content and each local crawler is assigned a broad content category as the target of its crawl. As pages are downloaded, a pretrained classifier is applied to each page, which determines a unique content category for it. Each local crawler operates independently unless it encounters a *boundary page*, a page not associated with its assigned category. Boundary pages are queued for transfer to their associated local crawlers on remote nodes. As boundary pages arrive from remote crawlers, they are treated as if they were directly downloaded by the local crawler.

The next section of the paper provides a review of related work. Section 3 discusses an approach to collaborative crawling that we view as the primary alternative to our topic-oriented approach. In this alternative, the Web is partitioned by hashing URLs and page contents. Sections 4 and 5 then examine issues in the design of the X4 crawler. In section 6, X4 is evaluated experimentally, including a comparison with hash-based partitioning.

2. RELATED RESEARCH

The most comprehensive study of Web crawlers and their design is Cho's 2001 Stanford Ph.D. thesis [10]. In particular, Chapter 3 of the thesis, along with a subsequent paper [11], represents the only study (we are aware of) that attempts to map and explore a full design space for parallel and distributed crawlers. Sharing many of our own goals, the work addresses issues of communication bandwidth, page quality and the division of work between local crawlers. As part of the work, Cho proposes and evaluates a hash-based approach similar to that discussed in the next section and suggests an occasional exchange of link information between nodes to improve the accuracy of locally computed page quality estimates. Cho's thesis does not directly address the issue of duplicate content detection in a distributed context.

Apart from Cho's work, most designs for parallel crawlers implement some form of global control, with the pending queue and related data structures maintained centrally and the actual download of pages taking place on worker nodes [4, 6]. One example is the WebFountain crawler [20], which is based on a cluster-of-workstations architecture. The software consists of three major components: *ants*, *duplicate detectors*, and a single system *controller*. The controller manages the system as a whole, maintaining global data structures and monitoring system performance. Ants are responsible for the actual retrieval of Web resources. The controller assigns each site to an ant for crawling, which retrieves all URLs from that site. The duplicate detectors recognize identical or near-identical content. A major feature of the WebFountain crawler is its maintenance of up-to-date copies of page contents by identifying those pages that change frequently and reloading them as needed.

The technical details of the crawlers used by commercial Web search services are naturally regarded as trade secrets, and there are few published details about their structure and implementation. One significant exception is Mercator, the crawler now used by the AltaVista search service (replacing the older Scooter crawler)[1]. Heydon and Najork [23, 24] describe in detail the problems associated with

[1] `http://www.research.digital.com/SRC/mercator`

creating a commercial-quality Web crawler and the solutions used to address these problems in Mercator. Written in Java, Mercator achieves both scalability and extensibility, largely through careful software engineering.

Research on *focused crawlers* is closely connected to the work described in the present paper. In essence, focused crawlers attempt to order the pending queue so that pages concerning a specific topic are placed earlier in the queue, with the goal of creating a specialized collection about this target topic. Chakrabarti et al. [9] introduce the notion of a focused crawler and experimentally evaluate an implementation of the concept using a set of fairly narrow topics such as "cycling", "mutual funds", and "HIV/AIDS". In order to determine the next URL to access, their implementation uses both a hypertext classifier [8], which determines the probability that a page is relevant to the topic, and a distiller, which identifies hub pages pointing to many topic-related pages [27].

Mukherjea [33] presents WTMS, a system for gathering and analyzing collections of related Web pages, and describes a focused crawler that forms a part of the system. The WTMS crawler uses a vector space similarity measure to compare downloaded pages with a target vector representing the desired topic; URLs from pages with higher similarity scores are placer earlier in the pending queue. Both McCallum et al. [30], and Diligenti et al. [18] recognize that the target pages of a focused crawl do not necessarily link directly to one another and describe focused crawlers that learn to identify apparently off-topic pages that reliably lead to on-topic pages. Menczer et al. [31] consider the problem of evaluating and comparing the effectiveness of the strategies used by focused crawlers.

The *intelligent crawler* proposed by Aggarwal et al. [1] generalizes the idea of a focused crawler, encompassing much of the prior research in this area. Their work assumes the existence of a predicate that determines membership in the target group. Starting at an arbitrary point, the crawler adaptively learns linkage structures in the Web that lead to target pages by considering a combination of features, including page content, patterns matched in the URL, and the ratios of linking and sibling pages that also satisfy the predicate.

3. HASH-BASED COLLABORATION

One approach to implementing a distributed crawler partitions the Web by computing hash functions over both URLs and page contents. When a local crawler extracts a URL from a retrieved page, its representation is first normalized by converting it to an absolute URL (if necessary) and then translating any escape sequences into their ASCII values. A hash function is then computed over the normalized URL, which assigns it to one of the n local crawlers. If the assigned local crawler is located on a remote node, the URL is transferred to that node. Once the URL is present on the correct node, it may be added to the node's pending queue. Similarly, each local crawler computes a hash function over the contents of each page that it downloads. This page-content hash function assigns the contents to one of the local crawlers, where duplicate detection and other post-processing takes place.

Under this hash-based scheme for collaboration, up to three local crawlers may be involved in processing each URL encountered: 1) the local crawler where it is encountered, 2) the node where it is assigned by the URL hash function, and 3) the node where the contents are assigned by the page-content hash function. Although many transfers between local crawlers cannot be avoided, local crawlers should maintain local tables of URLs and page contents that have been previously seen, to prevent unnecessary transfers

The URL hash function need not take the entire URL into account. For example, a URL hash function based only on the hostname ensures that all URLs from a given server are assigned to the same local crawler for download [10], allowing the load placed on the server to be better controlled. Similarly, the page-content hash function be based on normalized content, allowing near-duplicates to be assigned to the same node [5, 13].

The use of a page-content hash function may force considerable data transfer between local crawlers. With $n > 2$, a uniform hash function will map most retrieved pages to remote nodes. A downloaded page that is not mapped to the local node must be exported to its assigned node. In an n-node distributed crawler, the expected ratio of exported data to total data downloaded is $(n-1)/n$. As data is exported to remote nodes, data is imported from these nodes into the local crawler. If all local crawlers download data at the same rate, the expected ratio of imported data to total data downloaded is also $(n-1)/n$. In the limit, as the number of nodes is increased, the amount of data transferred between local crawlers is twice the amount of data downloaded from the Web.

A uniform hash function over the contents of a page will map it to a local crawler independently of the locations of pages that reference it. As the number of local crawlers increases, the probability that a referenced page will be assigned to the same node as its referencing page decreases proportionally. This property may have a negative impact on quality heuristics used to order the pending queue and, in turn, on the effectiveness of the crawl. Page quality heuristics that use backlink information may not have the information locally available to accurately estimate ordering metrics.

A solution proposed by Cho [10] is to have local crawlers periodically exchange backlink information. Cho demonstrates that a relatively small number of exchanges substantially improves local page quality estimates, but the approach increases the complexity of communication between local crawlers. To implement the approach, each local crawler must either selectively query the others for backlink information or transfer all backlink information to all other local crawlers.

An important parameter of a collaborative crawler is the probability p_l that linked page and its linking page will be assigned to the same node. In a hash-based collaborative crawler using a uniform hash function, $p_l = 1/n$. One goal of topic-oriented collaboration is to reduce the dependence of p_l on the number of nodes n.

4. TOPIC-ORIENTED COLLABORATION

In the previous section we assumed that the hash value for the contents of a specific page is independent of the hash value for pages that reference it. In this section we outline the design of a topic-oriented collaborative crawler that uses a text classifier to assign pages to nodes. Given the contents of a Web page, the classifier assigns the page to one of n distinct subject categories. Each subject category is associated with a local crawler. When the classifier assigns a page to a remote node, the local crawler transfers it to its assigned node for further processing. A topic-oriented collaborative crawler may be viewed as a set of broad-topic focused crawlers that partition the Web between them.

The breadth of the subject categories depends on the value of n. For n in the range 10-20, two of the subject categories might be BUSINESS and SPORTS. For larger n, the subject categories will be narrower, such as INVESTING, FOOTBALL and HOCKEY. The implementation of the classifier used in X4 will be discussed in the next section.

A potential advantage of replacing a simple page-content hash function with a text classifier is an increased likelihood that a linked page will be mapped to the same node as its linking page. For example, a link on a page classified as SPORTS may be more likely to reference another SPORTS page than a BUSINESS page. Many of the potential benefits of topic-oriented collaborative crawling derive from this assumption of *topic locality*, that pages tend to reference pages on the same general topic [17]. One immediate benefit of topic locality is a reduction in the bandwidth required to transfer pages from one local crawler to another. Only boundary pages, which are not assigned to the nodes that retrieved them, will be transferred. In addition, page quality metrics that depend on backlink information might be more accurately estimated when pages are grouped by assigned category, since more complete linkage information may be present.

As an additional advantage of topic-oriented collaboration, the output of each local crawler can be meaningfully regarded as an independent subcollection. Information brokers [21, 22] that weight the output of different search systems according to their expected performance on different query types may be able to take advantage of the topic focus of each subcollection.

The topic-oriented approach to collaborative crawling has the disadvantage that the same URL may be independently encountered and downloaded by multiple crawlers. In our approach, URLs are not hashed and are always retained on the nodes where they are encountered. If a URL is encountered by two or more nodes, each node will independently download the page, transferring it to a common node after categorization. While this property may appear to represent a serious limitation of topic-oriented collaborative crawling, we observe that a page will only be encountered by multiple nodes if pages from multiple categories reference it, a situation that the topic locality assumption tends to minimize.

The benefits of topic-oriented collaboration depend on the accuracy of the classifier and on the actual intra- and inter-category linkage patterns found in the Web. An experimental evaluation of these issues using the X4 crawler is reported in section 6. To provide context for this evaluation, we first describe and evaluate the simple classifier used in X4.

5. PAGE CATEGORIZATION

Text categorization is the grouping of text documents into a set of predefined *categories* or *topics*. Often, categorization is based on probabilities generated by training over a set of pre-classified documents containing examples from each topic. Document *features*, such as words, phrases and structural relationships, are extracted from the pre-classified documents and used to train the classifier. Given a unclassified document, the trained classifier extracts features from the document and assigns the highest-probability category to it.

Text categorization is a heavily studied subject. A variety of machine learning and information retrieval techniques have been applied to the problem, including Rocchio feedback [25], support vector machines [26], expectation-maximization [34], and boosting [35]. Yang and Liu [37] provide a recent comparison of five widely used methods.

Many these techniques have been applied to categorize Web pages and in some cases have been extended to exploit the unique properties of Web data. Much of this work has been in the context of focus crawlers, discussed in section 2. Chakrabarti et al. [8] take advantage of the Web's link structure to improve Web page categorization. Using an iterative relaxation labeling approach, pages are classified by using the categories assigned to neighboring pages as part of the feature set. Dumais and Chen [19] take advantage of the large collections of hierarchically organized Web pages provided by organizations such as Yahoo! and LookSmart to develop a hierarchical categorization technique based on support vector machines.

To select a page categorization technique for use in X4, several attributes of the available techniques were considered. First, categorization should be based only on document contents. If the contents of neighboring documents are considered by the classifier, a page's neighborhood would have to be retrieved by each crawler encountering it. Retrieving the neighborhood of a page before it is classified is likely to increase the overlap between crawlers, something that X4 seeks to avoid. Second, after training is completed the classifier must remain static and cannot learn from new data, since the category assigned to a page's contents by every local crawler must be the same. Third, the classifier should be efficient enough to categorize pages at the rate they are downloaded. Crawling a major portion of the Web requires a minimum download rate of several Mbps, and the classifier should be able to match this rate without requiring more resources than the crawler itself. Finally, the *accuracy* of the classifier, the percent of pages correctly classified, should be as high as possible. Moreover, to minimize the number of boundary pages encountered, the probability that a linked page is classified in the same category as its linking page, the topic locality, should also be as high as possible.

Of the many available techniques, we choose three for further study on the basis of their simplicity and potential for efficient implementation: a basic Naive Bayes classifier [32], a classifier based on Rocchio relevance feedback [25], and a probabilistic classifier due to Lewis [29].

Data from the Open Directory Project[2] (ODP) was used to train and test the classifiers. The ODP is a self-regulated organization maintained by volunteer experts who categorize URLs into a hierarchical class directory, similar to the directories provided by Yahoo! and others. At the top level, there are 17 categories. Volunteers examine the contents of each URL to determine its category. Each level in the hierarchy contains a list of relevant external links and a list of links to subcategories.

A snapshot of 673MB of data was obtained from the ODP. For the purpose of our experiments, the entire URL directory tree was collapsed into the top categories. Two categories were given special treatment. The REGIONAL cate-

[2] `http://www.dmoz.org`

ADULT	ARTS	BUSINESS
COMPUTERS	GAMES	HEALTH
HOME	KIDS/TEENS	NEWS
RECREATION	REFERENCE	SCIENCE
SHOPPING	SOCIETY	SPORTS
	WORLD	

Figure 1: ODP categories used in the topic classifier.

gory, which encompasses pages specific to various geographical areas, was eliminated entirely, since we believe it represents not so much a separate topic as an alternative organization. Pages in the WORLD category, which are written in languages other than English, were also ignored during the initial classifier selection phase. In the final X4 classifier non-English pages are handled by a separate language identifier. The 16 top-level categories (including WORLD but excluding REGIONAL) are listed in figure 1. Ultimately, these became the target categories used by the X4 classifier.

For categorization, pages are preprocessed by first removing tags, scripts, and numerical information. The remaining text is tokenized into terms based on whitespace and punctuation, and the terms are converted to lower case. The resulting terms are treated as the document features required by the classifiers.

From each ODP category, 500 documents were randomly selected for training and 500 for testing. Only HTML documents containing more than 50 words after preprocessing were considered as candidates for selection. The performance of the classifiers is shown in figure 2. Overall, the Naive Bayes classifier achieved an accuracy of 60.7%, the Lewis classifier 58.2%, and the Rocchio-TFIDF classifier 43.7%.

For topic-oriented crawling a more important statistic is the *topic locality*, which we measure by the proportion of linked pages that are classified into the same category as their linking page. To test topic locality, 100 Web pages from each category were randomly selected from the training data. Five random links from each page (or all links if the page did not have five) were retrieved and classified. The results are shown in figure 3. Overall, the Naive Bayes classifier achieved a topic locality of 62.4%, the Lewis classifier 62.3%, and the Rocchio-TFIDF classifier 48.9%.

To test the speed of the classifiers, 30,000 Web pages with an average length of 7,738 bytes were fed into each. The Naive Bayes classifier achieved a throughput of 19.50 pages/second, the Lewis classifier 2.89 pages/second, and the Rocchio-TFIDF classifier 1.61 pages/second. Although the Naive Bayes and Lewis classifiers perform comparably in terms of categorization accuracy, outperforming the Rocchio-TFIDF classifier, the six times greater speed of the Naive Bayes classifier recommended its use in the X4 crawler.

Before topic categorization, the natural language in which a page is written must be determined. Language-specific classifiers may only be used after language identification to partition the pages into two or more language-specific topics. The division of nodes between natural languages and language-specific topics depends on the number of local crawlers required and mix of Web resources targeted by the crawl. For the purpose of the experiments reported in this paper, we group all non-English pages into a single category WORLD, following the ODP organization.

A number of simple statistical language identification techniques have shown good performance [3, 7, 15]. For X4, we identified English-language pages by the proportion of common English words appearing in them. To train and test the identifier, we selected 500 random pages from each category, including WORLD. The resulting language identifier has an accuracy of 96.1% and 90.2% on identifying English and non-English pages respectively.

6. EXPERIMENTAL EVALUATION

The X4 crawler is implemented as an extension to the MultiText Web Crawler, which was originally developed as part of the MultiText project at the University of Waterloo to gather data for Web-based question answering [14]. The crawler has since been used by a number of external groups. Sharing design goals with Mercator [23], the MultiText crawler is designed to be highly modular and configurable, and has been used to generate collections over 1TB in size. On an ordinary PC, the crawler can maintain download rates of millions of pages a day, including pre- and post-processing of the pages. The core of the crawler provides a dataflow scripting language that coordinates the activities of independent software components, which perform the actual operations of the crawl. Each individual component is responsible for a specific crawling task such as address resolution, page download, URL extraction, URL filtering, and duplicate content handling. The core of the crawler also provides transaction support, allowing crawler actions to be rolled back and restarted after a system failure.

X4 was created by adding two new components to the MultiText crawler. One component is the topic classifier; the other other component is a data transfer utility. Data to be transferred to remote nodes is queued locally by the topic classifier. Periodically (every 30 minutes) the data transfer utility polls remote nodes and downloads any queued data; `scp` is used to perform the actual transfer. Apart from changes to our standard crawl script to add calls to the classifier and data transfer utility, no other changes to the MultiText Crawler were required.

For our experimental evaluation, the X4 pending queue was maintained in breadth-first order with one exception. If a breadth-first ordering would place an excessive load on a single host, defined as more than 0.2% of total crawler activity over a time period of roughly one hour, URLs associated with that host were removed and requeued until the anticipated load dropped to an acceptable level.

For our experimental evaluation of X4, we used the Naive Bayes topic classifier trained on ODP data, described in the previous section. After preprocessing, each retrieved page was first checked to determine if it was *unclassifiable*, which we defined as containing fewer than 50 terms after the removal of tags and scripts during preprocessing. We arbitrarily assigned unclassifiable pages to category #0 (ADULT). Unless it was unclassifiable, each page then had the language identifier applied to it. If the page was not assigned to the WORLD category by the language identifier, the topic classifier was applied to assign the page to its final category.

In our experiments, a local crawler was associated with each the 16 categories of figure 1. These 16 local crawlers were mapped onto six nodes of a workstation cluster by assigning three local crawlers to five of the nodes and a single

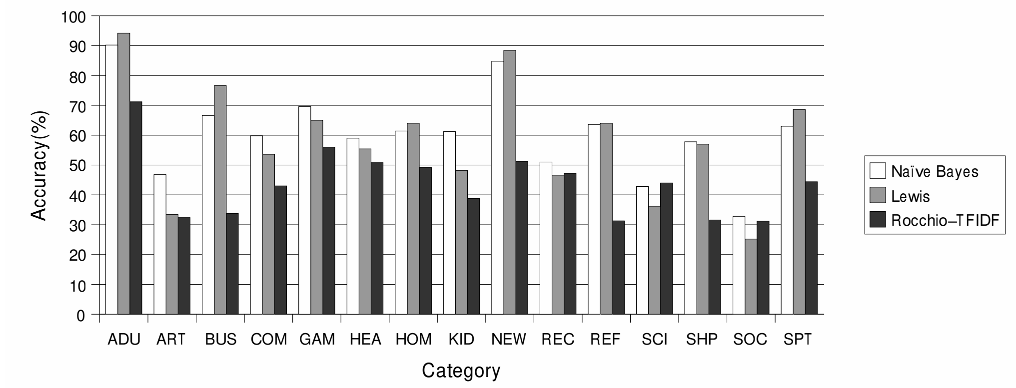

Figure 2: Accuracy of classifiers (ODP data).

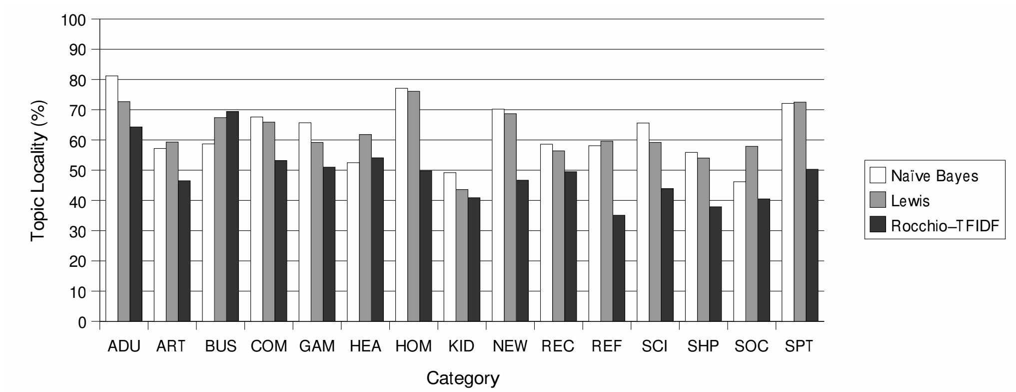

Figure 3: Topic locality of classifiers (ODP data).

category (WORLD) to one of the nodes. Although multiple local crawlers were assigned to the same node for testing purposes, these local crawlers acted in all respects as if they were executing on distinct nodes.

We generated a 137GB experimental crawl in June 2001. Due to limitations on our available bandwidth to the general Internet, the crawlers as a group were limited to a download rate of 256KB/second, with each local crawler limited to 16KB/second. Due to the difficulty of controlling the download rate at such low speeds, the actual download rates varied from 11 KB/second to 16 KB/second. In total, 8.9 million pages were downloaded and classified. Figure 4 plots the distribution of the data across the local crawlers, showing both the volume of downloaded data retained locally after categorization and the volume of data imported from other local crawlers.

The topic locality achieved by each local crawler is plotted in figure 5. Generally the topic locality achieved during the crawl was slightly lower than that seen on the ODP data (figure 3), but the relative topic locality achieved for each topic was roughly the same. The main exception is local crawler #0 (ADULT), where the topic locality was affected by our arbitrary decision to assign unclassifiable pages to its associated category. For comparison, the equivalent value for hash-based collaboration (p_l) is 1/16 or 6.25%.

With hash-based collaborative crawling, URLs are hashed and exchanged, mapping each URL to a unique local crawler and preventing multiple crawlers from downloading the same URL. This is not the case with topic-oriented collaborative crawling. Since only page contents are exchanged during topic-oriented collaboration, multiple crawlers will download a URL when that URL is referenced by pages from more than one category. For $1 \leq i \leq 16$, figure 6 shows the log of the number of pages retrieved by exactly i local crawlers. The number of pages decreases rapidly as i increases. Only 654 pages were retrieved by all 16 local crawlers, generally the home pages of major organizations or products, such as `www.nytimes.com` and `www.microsoft.com/ie`.

In order to examine the properties of the subcollections generated by the local crawlers, we ordered the pages in each subcollection using the pagerank algorithm ($d = 0.85$). To permit a direct comparison with hash-based collaboration, we redistributed the pages into 16 different subcollections using a page-content hash function and computed the pagerank ordering of each. Finally, we gathered all of the pages into a single collection and computed a global pagerank ordering.

In figure 7 we compare the pagerank ordering computed over each subcollection with the global pagerank ordering computed over the combined collection. For each subcollection, the figure reports the Kendall τ rank correlation between the pagerank ordering computed over the subcol-

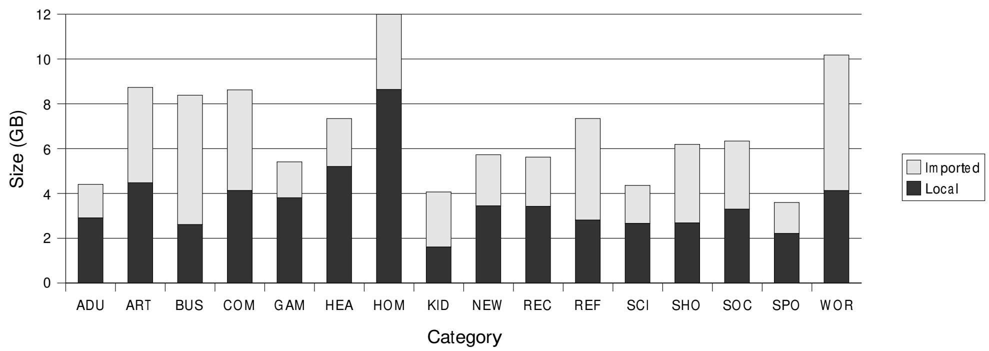

Figure 4: Distribution of crawled data.

lection and the pagerank ordering of the same pages computed over the global collection. A higher correlation coefficient indicates a more accurate local estimate of the global pagerank ordering. As might be expected from the use of a uniform hash function, the coefficients the hash-based subcollections are nearly identical. In all cases, the coefficients for the topic-oriented subcollections are greater than the coefficients for the hash-based subcollections.

7. CONCLUSIONS AND FUTURE WORK

In this paper we propose the concept of topic-oriented collaborative crawling as a method for implementing a general-purpose distributed Web crawler and demonstrate its feasibility through an implementation and experimental evaluation. In contrast with the URL- and host-based hashing approach evaluated by Cho [10,11], the approach allows duplicate page content to be recognized and generates subcollections of pages that are related by topic, rather than location.

X4 could be extended and improved in several ways. In the current implementation, pending queues are maintained in breadth-first order. Instead, a focused crawling technique might be used to order the pending queues, placing URLs that are more likely to reference on-topic resources closer to the front of the queue. Such a technique could substantially increase the proportion of on-topic pages retrieved and decrease the number of transfers between local crawlers.

X4 transfers the contents of each boundary page from the node that retrieves it to the node where the classifier assigns it. This design decision was taken as a consequence of our desire to map the contents of each page uniquely to a local crawler in order to facilitate duplicate detection and other post-processing. An alternative design would be to transfer only the URLS of links appearing on boundary pages. The contents of boundary pages would not be transferred. A disadvantage of this approach is that the contents of each URL is not uniquely associated with a local crawler. Every local crawler that encounters the URL as a boundary page will retain a copy of its contents. An advantage of the approach is that the communication overhead between local crawlers is further reduced. The only communication between local crawlers is the transfer of URLs extracted from boundary pages. The retention of boundary page contents on multiple nodes may have advantages of its own. Topic locality implies that these pages are likely to be related to the topics of the local crawlers where they are encountered, and the high backlink count associated with a boundary page encountered by many crawlers implies high quality.

In the current work, our primary interest was not Web page categorization itself, and other text categorization methods could be explored for use in X4. Techniques such as feature selection [36] might be used to improve both efficiency and accuracy. Problems associated with incremental crawling and dynamically changing content were not considered and should be examined by future work. Our evaluation is based on a relatively small crawl (137GB), and a more thorough evaluation based on a multi-terabyte crawl might reveal issues that are not obvious from our current experiments. Finally, X4 should be tested for its ability to scale to a larger number of nodes, with a correspondingly larger number of categories.

8. REFERENCES

[1] Charu C Aggarwal, Fatima Al-Garawi, and Philip S. Yu. Intelligent crawling on the World Wide Web with arbitrary predicates. In *Tenth International World Wide Web Conference*, pages 96–105, May 2001.

[2] Brian Amento, Loren Terveen, and Will Hill. Does "authority" mean quality: Predicting expert quality ratings of Web documents. In *23rd Annual International ACM SIGIR Conference on Research and Developement in Information Retrieval*, pages 296–303, 2000.

[3] Dario Benedetto, Emanuele Caglioti, and Vittorio Loreto. Language trees and zipping. *Physical Review Letters*, 88(4), January 2002.

[4] Sergey Brin and Lawrence Page. The anatomy of a large-scale hypertextual Web search engine. In *Seventh International World Wide Web Conference*, pages 107–117, April 1998.

[5] Andrei Z. Broder, Steven C. Glassman, Mark S. Manasse, and Geoffrey Zweig. Syntactic clustering of the Web. In *Sixth International World Wide Web Conference*, 1997.

[6] Mike Burner. Crawling towards Eternity. *Web Techniques*, 2(5), May 1997.

[7] W. Cavnar and J. Trenkle. N-gram based text categorization. In *3rd Annual Symposium on Document Analysis and Information Retrieval*, pages 161–175, 1994.

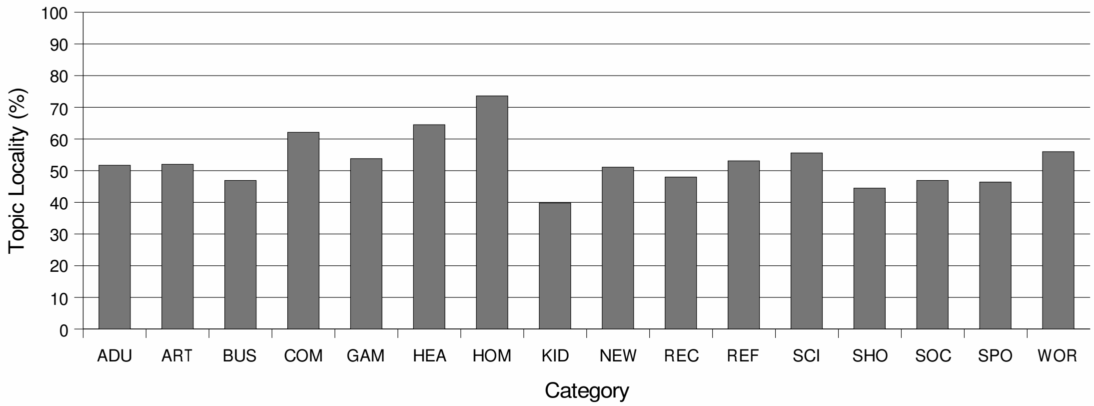

Figure 5: Topic locality of crawled data.

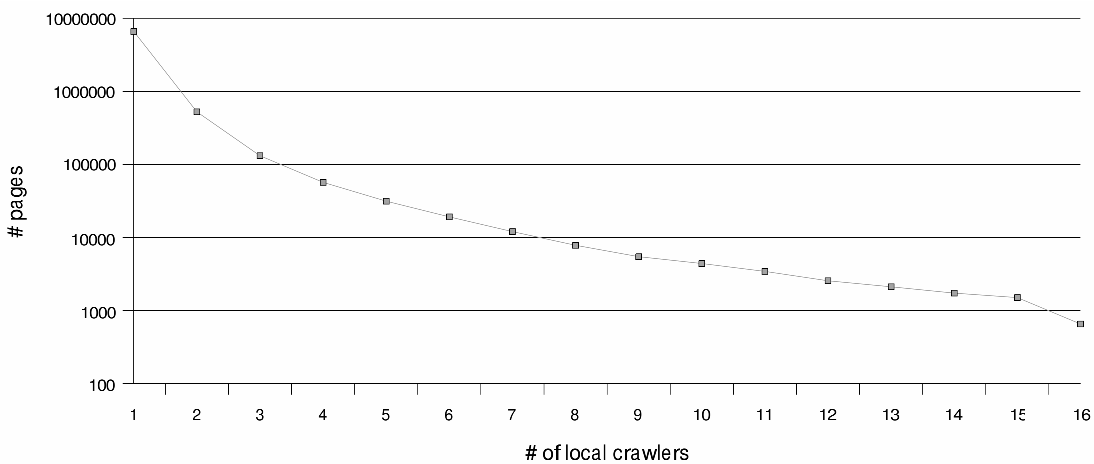

Figure 6: Pages downloaded by multiple local crawlers.

node #	0	1	2	3	4	5	6	7
topic-oriented	0.657	0.708	0.646	0.702	0.806	0.840	0.594	0.649
hash-based	0.525	0.526	0.524	0.524	0.528	0.526	0.524	0.524
node #	8	9	10	11	12	13	14	15
topic-oriented	0.724	0.738	0.749	0.786	0.636	0.674	0.706	0.716
hash-based	0.528	0.523	0.526	0.523	0.524	0.523	0.524	0.525

Figure 7: Correlation of subcollection pageranks with global pagerank.

[8] Soumen Chakrabarti, Byron Dom, and Piotr Indyk. Enhanced hypertext categorization using hyperlinks. In *ACM SIGMOD International Conference on Management of Data*, pages 307–318, 1998.

[9] Soumen Chakrabarti, Martin van den Burg, and Byron Dom. Focused crawling: A new approach to topic-specific Web resource discovery. In *Eighth International World Wide Web Conference*, pages 545–562, May 1999.

[10] Junghoo Cho. *Crawling the Web: Discovery and Maintenance of Large-Scale Web Data*. PhD thesis, Stanford University, 2001.

[11] Junghoo Cho and Hector Garcia-Molina. Parallel crawling. In *Eleventh International World Wide Web Conference*, May 2002.

[12] Junghoo Cho, Hector Garcia-Molina, and Lawrence Page. Efficient crawling through URL ordering. In *Seventh International World Wide Web Conference*, April 1998.

[13] Junghoo Cho, Narayanan Shivakumar, and Hector Garcia-Molina. Finding replicated Web collections. In *ACM SIGMOD International Conference on Management of Data*, pages 355–366, 2000.

[14] Charles L. A. Clarke, Gordon V. Cormack, and Thomas R. Lynam. Exploiting redundancy in question answering. In *24th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 358–365, September 2001.

[15] Jim Cowie, Yevgeny Ludovik, and Ron Zacharski. An autonomous, Web-based, multilingual corpus collection tool. In *International Conference on Natural Language Processing and Industrial Applications*, 1998.

[16] Mark Craven, Dan DiPasquo, Dayne Freitag, Andrew McCallum, Tom Mitchell, Kamal Nigan, and Séan Slattery. Learning to construct knowledge bases from the World Wide Web. *Artificial Intelligence*, 118(1–2):69–113, 2000.

[17] Brian D. Davison. Topical locality on the Web. In *23rd Annual International ACM SIGIR Conference on Research and Developement in Information Retrieval*, pages 272–279, 2000.

[18] M. Diligenti, F. M. Coetzee, S. Lawrence, C. L. Giles, and M. Gori. Focused crawling using context graphs. In *26th International Conference on Very Large Databases*, pages 527–534, September 2000.

[19] Susan T. Dumais and Hao Chen. Hierarchical classification of Web content. In *23rd Annual International ACM SIGIR Conference on Research and Developement in Information Retrieval*, pages 256–263, 2000.

[20] Jenny Edwards, Kevin McCurley, and John Tomlin. An adaptive model of optimizing performance of an incremental Web crawler. In *Tenth International World Wide Web Conference*, pages 106–113, May 2001.

[21] Susan Gauch, Guijun Wang, and Mario Gomez. ProFusion:Intelligent fusion from multiple, distributed search engines. *Journal of Universal Computer Science*, 2(9), September 1996.

[22] David Hawking and Paul Thistlewaite. Methods for information server selection. *ACM Transactions on Information Systems*, 17(1):40–76, January 1999.

[23] Allan Heydon and Marc Najork. Mercator: A scalable, extensible Web crawler. *World Wide Web*, 1(2):219–229, December 1999.

[24] Allan Heydon and Marc Najork. Performance limitations of the Java Core libraries. In *ACM 1999 Java Grande Conference*, pages 35–41, June 1999.

[25] Thorsten Joachims. A probabilistic analysis of the Rocchio algorithm with TFIDF for text classification. In *14th International Conference on Machine Learning*, pages 143–151, 1997.

[26] Thorsten Joachims. A statistical learning model of text classification for support vector machines. In *24th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 128–136, September 2001.

[27] Jon M. Kleinberg. Authoritative sources in a hyperlinked environment. *Journal of the ACM*, 46(5):604–632, 1999.

[28] Cody C. T. Kwok, Oren Etzioni, and Daniel S Weld. Scaling question answering to the Web. In *Tenth International World Wide Web Conference*, pages 150–161, May 2001.

[29] D. Lewis. An evaluation of phrasal and clustered representations on a text categorization task. In *15th Annual International ACM SIGIR Conference on Research and Developement in Information Retrieval*, pages 37–50, 1992.

[30] Andrew McCallum, Kamal Nigam, Jason Rennie, and Kristie Seymore. Building domain-specific search engines with machine learning techniques. In *AAAI-99 Spring Symposium on Intelligent Agents in Cyberspace*, 1999.

[31] Filippo Menczer, Gautam Pant, Padmini Srinivasan, and Miguel E. Ruiz. Evaluating topic-driven Web crawlers. In *24th Annual International ACM SIGIR Conference on Research and Developement in Information Retrieval*, pages 241–249, September 2001.

[32] Tom Mitchell. *Machine Learning*. McGraw Hill, 1997.

[33] Sougata Mukherjea. WTMS: A system for collecting and analyzing topic-specific Web information. In *Ninth International World Wide Web Conference*, May 2000.

[34] Kamal Nigam, Andrew Kachites McCallum, Sebastian Thrun, and Tom Mitchell. Text classification from labeled and unlabeled documents using EM. *Machine Learning*, 39(2/3):103–134, 2000.

[35] Robert E. Schapire, Yoram Singer, and Amit Singhal. Boosting and Rocchio applied to text filtering. In *21st Annual International ACM SIGIR Conference on Research and Developement in Information Retrieval*, pages 215–223, August 1998.

[36] Vaughan Shanks and Hugh E. Williams. Fast categorisation of large document collections. In *8th International Symposium on String Processing and Information Retrieval*, pages 194–204, November 2001.

[37] Yiming Yang and Xin Liu. A re-examination of text categorization methods. In *22th Annual International ACM SIGIR Conference on Research and Developement in Information Retrieval*, pages 42–49, August 1999.